Decidability results for primitive recursive algorithms

René David

Decidability results for primitive recursive algorithms

R. David*

January 30, 2002

Abstract

In this paper I use the notion of trace defined in [9] to extend T.Coquand's *constructive* proof [6] of the ultimate obstination theorem of L. Colson to the case when mutual recursion is allowed. As a by product I get an algorithm that computes the value of a primitive recursive combinator applied to lazy integers (infinite or partially undefined arguments may appear). I also get, as T. Coquand got from his proof, that, even when mutual recursion is allowed, there is no primitive recursive definition f such that $f(S^n(\perp)) = S^{n^2}(\perp)$.

1 Introduction

In [3], Colson proved the so-called "ultimate obstination theorem". This theorem asserts that a primitive recursive algorithm always reads, and so locks, on a particular input argument to complete its computation. This behaviour does not allow the computation to shift from one argument to another one in order to efficiently compute a function, as for example the *inf* function studied originally by Colson. In [9], by using the syntactic notion of trace which is a simplified version of the sequential algorithms of Berry and Curien (see [2] or [1], chapter 14) a reformulation of this theorem is given and proved.

The proof of this theorem, as given in [3] or [9], is not constructive : it does not give a way to determine on which particular input argument the algorithm locks.

In this paper, I show that this can be done in a constructive way. This was proved by T.Coquand in [6] for primitive recursive algorithms. This is extended here in the case where mutual or alternate recursion is allowed. Alternate recursion (see definition 2) has been introduced by P. Valarcher [17] to give a good algorithm to compute the *inf* function.

The proof of the main theorem needs a difficult combinatorial result (proposition 18) which has some interest by itself.

Decidability results : I show that, even when mutual or alternate recursion is allowed, various problems are decidable. For example, it is possible to compute the intentional behaviour of a primitive recursive combinator f, i.e. the value of f when applied to lazy integers (infinite or partially undefined arguments may appear). Note that, when alternate recursion is allowed, the ultimate obstination theorem obviously fails. This shows that the decidability results have nothing to do with ultimate obstination ! I also show that when lists or change of parameters in the recursion scheme are allowed, these problems become undecidable.

As a by-product of the proof I also get :

*Laboratoire de Mathématiques. Campus Scientifique. 73376 Le Bourget du Lac Cedex. email david@univ-savoie.fr

The Input-Output behaviour of primitive recursive combinators : I show, for example, that, even when mutual or alternate recursion is allowed, there is no primitive recursive definition f such that $f(S^n(\bot)) = S^{n^2}(\bot)$. This was proved by T. Coquand in the ordinary case. I believe the same technic *could* imply other similar results such as, for example, if mutual recursion is restricted to at most two functions, there is no f such that $f(S^n(\bot)) = S^{[n/3]}(\bot)$ where where $[x]$ is the integer part of x but this seems to need a combinatorial lemma that I have not been able to prove (nor disprove).

I recall here after the main intuitions concerning the notion of trace. Let $\mathcal{N}$ be the domain of lazy integers. An element e of $\mathcal{N}$ can be seen as a partial function that *fills* some accessible *cells* (in the sense of [2]) with the constructors S, 0 and $\bot$. Since $\bot$ corresponds to a lack of information, a cell filled with $\bot$ is often said to be *unfilled.* For example (see figure 1) in $e_0 = S(0)$ the accessible cells are the ones denoted by their address 0 and 1. The first one is filled with S and the second with 0. In $e_1 = S^2(\bot)$ the accessible cells are the ones denoted $0, 1, 2$. The cells 0 and 1 are filled with S and the third one is unfilled.

$e_0 =$

cell number	0	1
constructor	S	0

and $e_1 =$

cell number	0	1	2
constructor	S	S	$\bot$

fig. 1

The set of *traces* is defined as follows. Let W be the set of (finite or infinite) words on the alphabet $\{x_n \ / \ n \geq 0, \ x \text{ is a letter}\}$. A trace is a pair (e, λ) where $e \in \mathcal{N}$ and λ is a labelling i.e. a function from the accessible cells of e to W (see examples in figure 2).

To each primitive recursive definition (*prc*) f we associate a function $[[f]]$ from traces to traces which "codes" the way f gets its result : the fact that the *token* x_i occurs in $\lambda(n)$ intuitively means that the cell i of the element named x has been used to get $e(n)$.

An example is given in figure 2 : define *add* as usual by $add(0, m) = m$ and $add(Sn, m) = S \ add(n, m)$.

- The trace t_2 means that to get S the algorithm has used the cell 0 of t_0 and to get 0 the algorithm has used first the cell 1 of t_0 and next the cell 0 of t_1.

- The trace t_3 means that to get S the algorithm has used first the cell 0 of t_1 and next the cell 0 of t_0 and to get 0 the algorithm has used the cell 1 of t_0.

$t_0 =$

cell number	0	1
constructor	S	0
labelling	x_0	x_1

$t_1 =$

cell number	0
constructor	0
labelling	y_0

$t_2 = [[add]](t_0, t_1) =$

cell number	0	1
constructor	S	0
labelling	x_0	$x_1 \ y_0$

$t_3 = [[add]](t_1, t_0) =$

cell number	0	1
constructor	S	0
labelling	$y_0 \ x_0$	x_1

fig. 2

Since a trace carries the informations on a computation and not only on the result, this notion allows to also *compose the computations.* I believe it also makes

the proofs easier and, at least, closer to the intuition than in the original formulation of Colson. In particular, the extension of Coquand's constructive result to the case where mutual recursion is allowed would probably be impossible without the notion of trace.

This notion of trace is related to the sequential algorithms introduced by Berry and Curien ([2] or [1], chapter 14) as follows. In their terminology, a sequential algorithm is a tree. Each branch of this tree corresponds to the computation of the algorithm on particular arguments, that is exactly (with a slight variation on the syntax and the terminology) what I call a trace. In particular $[[f]]$ can be seen as the sequential algorithm associated to f.

The paper is organized as follows : I recall the notion of trace (section 2) and its main properties (section 5). The main result is given in section 3. In section 4, I give a combinatorial result that is crucial for the proof of the main theorem. The sections 6, 7 and 8 are devoted to its proof. In section 9, I give the undecidability results. Finally the appendix gives the proof of the combinatorial proposition of section 4.

2 The trace

In this section I recall, for self completeness, the main definitions about traces. More details can be found in [9]. Since, in this paper, the only data type I am concerned with, is the data type of integers many things are simpler than in the general case. I thus adapt the definitions of [9] to this case.

Definition 1 *1. The scheme for primitive recursion is :* $f(0, \overrightarrow{r}) = g(\overrightarrow{r})$ *and* $f(Sx, \overrightarrow{r}) = h(f(x, \overrightarrow{r}), x, \overrightarrow{r})$.

2. The scheme for mutual recursion is : $f_i(0, \overrightarrow{r}) = g_i(\overrightarrow{r})$ *and* $f_i(Sx, \overrightarrow{r}) = h_i(f_1(x, \overrightarrow{r}), ..., f_k(x, \overrightarrow{r}), x, \overrightarrow{r})$.

3. The scheme for alternate recursion is :

$f(0, y_2, ..., y_k, \overrightarrow{r}) = g_1(y_2, ..., y_k, \overrightarrow{r})$

$f(Sy_1, 0, y_3..., y_k, \overrightarrow{r}) = g_2(y_1, y_3, ..., y_k, \overrightarrow{r})$

$f(Sy_1, Sy_2, 0, ..., y_k, \overrightarrow{r}) = g_3(y_1, y_2, ..., y_k, \overrightarrow{r})$

...

$f(Sy_1, ..., Sy_{k-1}, 0, \overrightarrow{r}) = g_k(y_1, ..., y_{k-1}, \overrightarrow{r})$

$f(Sy_1, ..., Sy_k, \overrightarrow{r}) = h(f(y_1, ..., y_k, \overrightarrow{r}), y_1, ..., y_k, \overrightarrow{r})$

Definition 2 *1. The sets of prc (primitive recursive combinators) are defined as the least sets containing the projections, the constructors* S *and* 0 *and which are closed under composition and primitive recursion.*

2. The sets of prc_{mut} *are defined in the same way as prc but definition by mutual recursion is allowed.*

3. The sets of prc_{alt} *are defined in the same way as prc but definition by alternate recursion is allowed.*

Examples and comments

1. For the simplicity of notations, I assume, without loss of generality, that the recursion always is on the first argument of the *prc*.

2. The addition is defined by : $add(0,n) = n$ and $add(Sm,n) = S\ add(m,n)$. Thus *add* is a *prc*.

3. The functions *odd* and *even* are defined by : $even(0) = 1$ and $odd(0) = 0$. $even(Sx) = odd(x)$ and $odd(Sx) = even(x)$. Thus *odd* and *even* are in prc_{mut}.

4. The function *inf* is defined by : $inf(0,0) = 0$, $inf(Sx,0) = 0$ and $inf(Sx,Sy) = Sinf(x,y)$. Thus *inf* is in prc_{alt} and it is easy to see that the computation time of $inf(S^n(0),S^m(0))$ is $inf(n,m)$.

5. It is well known that, as functions, the sets *prc*, prc_{mut} and prc_{alt} are equal but this paper shows, in particular, that, as algorithms, they are not.

Definition 3 *1. N (resp. N^*, Z) is the set of non negative (resp. positive, negative or non negative) integers.*

2. An element e of $\mathcal{N}$ is a partial function from an initial segment of N (denoted by $Acc(e)$) into $\{S, 0, \perp\}$ satisfying :

- $0 \in Acc(e)$
- *If* $(n+1) \in Acc(e)$, *then* $e(n) = S$.
- *If* $e(n) = \perp$, *then* $(n+1) \notin Acc(e)$.

3. An element e is finite iff $Acc(e)$ is finite.

4. Let e, e' be elements of $\mathcal{N}$. $e \leq e'$ means : $Acc(e) \subseteq Acc(e')$ and for all $n \in Acc(e)$, if $e(n) \neq \perp$, then $e(n) = e'(n)$.

5. I will denote the elements of $\mathcal{N}$ as :

- $S^n(0) = \{(i,S) \ / \ 0 \leq i < n\} \cup \{(n,0)\}$,
- $S^n(\perp) = \{(i,S) \ / \ 0 \leq i < n\} \cup \{(n,\perp)\}$
- $S^\omega = \{(i,S) \ / \ i \in N\}$.

Comment

$Acc(e)$ represents the set of integers that are *accessible* in e. In this presentation, this simply is the domain of e. In the general case (when various data types are allowed) it was more convenient to define two distinct sets : the domain of e and $Acc(e)$. I have kept the notation $Acc(e)$ to remain compatible with the notations of [9].

Definition 4 *1. Let $\Sigma = \{x_n \ / \ x$ is a letter and $n \in N\}$. The elements of Σ are called tokens.*

2. A word is a finite (possibly empty) or infinite sequence of tokens. The empty word is denoted by ε. The set of words is denoted by W.

3. Let u, u' be words. $u \leq u'$ means that u is a prefix of u' and $u \uparrow p$ denotes, for $p \leq lg(u)$, the prefix of u of length p.

4. $u + u'$ is the result of concatenating u' at the end of u. More generally, if (u_k) is a (finite or infinite) sequence of words $u_0 + u_1 + ...$ will be denoted by $\sum u_k$.

5. Let u_n be a sequence of words. Say that $u_n \to u$ if for each p there is an n_0 such that for all $n \geq n_0$, $u_n \uparrow p = u \uparrow p$. This unique u is denoted by $Lim(u_n)$.

Definition 5 *1. A trace is a pair (e, λ) where e is an element of $\mathcal{N}$ and λ is a labelling function $\lambda : Acc(e) \to W$ such that : $\forall n \in Acc(e)$, if $e(n) \neq \perp$, then $\lambda(n)$ is finite.*

2. *A trace (e, λ) is finite if e is finite and all labels are finite.*

3. *The ordering on traces is given by : $(e, \lambda) \leq (e', \lambda')$ iff $e \leq e'$ and for each n in $Acc(e)$, $\lambda(n) \leq \lambda'(n)$ and, if $e(n) \neq \perp$, then $\lambda(n) = \lambda'(n)$.*

4. *The set of traces is denoted by T.*

5. *Let e be an element of $\mathcal{N}$ and x be a letter. The trace (e, λ) where $\lambda(n) = x_n$ for all $n \in Acc(e)$ will be denoted as $e[x]$. A trace as $e[x]$ is called an element named x.*

6. *Let $t = (e, \lambda)$ be a trace. e is called the value of t and is denoted by $Val(t)$. λ is called the labelling of t and is denoted by $Lab(t)$.*

Proposition 6 *T with its ordering forms a domain. In particular :*

1. *Every trace is a least upper bound (denoted by Sup) of an increasing sequence of finite traces.*

2. *Every increasing sequence has a Sup.*

The following notations will be convenient at many places.

Definition 7 *Let $t = (e, \lambda)$ be a trace and w be a finite word.*

1. *$w + t$ is the trace (e, λ') defined by : $\lambda'(0) = w + \lambda(0)$ and $\lambda'(n) = \lambda(n)$ for $n \geq 1$.*

2. *$\langle (S, w)\ t \rangle$ (or simply $(S, w)\ t$ if no confusion is possible) is the trace (e', λ') defined by : $e'(0) = S$, $\lambda'(0) = w$ and, for $n \geq 0$, $e'(n+1) = e(n)$ and $\lambda'(n+1) = \lambda(n)$.*

3. *Let x be a letter. $t\langle x + k \rangle = (e, \lambda')$ where λ' is obtained from λ by replacing x_j by x_{j+k} for all j.*

Example

- $y_0 + S(0)[x] = (S, y_0\ x_0)\ (0, x_1)$
- $S^n(\perp)[x] = (S, x_0)(S, x_1)...(S, x_{n-1})(\perp, x_n)$.
- Let $t = S^\omega[x]$, then $t = \langle (S, x_0)\ s \rangle$ where $s = t\langle x + 1 \rangle$.

Definition 8 *Let f be a function from T^n to T.*

1. *f is increasing if for all $t_j \leq t'_j$, $f(t_1, ..., t_k) \leq f(t'_1, ..., t'_k)$.*

2. *f is continuous if it is increasing and preserves the Sup of increasing sequences.*

Proposition 9 *Every n-ary $f \in prc$ (resp. prc_{mut}, resp. prc_{alt}) induces (in a unique way) a continuous function (denoted by $[[f]]$) from T^n to T such that :*

- $[[0]](t_1, ..., t_n) = (0, \varepsilon)$
- $[[S]](t) = (S, \varepsilon)\ t$
- *If f is the i-th projection then $[[f]](t_1, ..., t_n) = t_i$*

- *If* $f = g(h_1, ..., h_k)$ *then,*

$$[[f]](t_1,\ ...,\ t_n) = [[g]](r_1,\ ..., r_k) \text{ where } r_j = [[h_j]](t_1,\ ...,\ t_n)$$

- *If* f *is defined by ordinary or mutual recursion and the recursive equations are* $f_i(0, \overrightarrow{s}) = g_i(\overrightarrow{s})$ *and* $f_i(Sx, \overrightarrow{s}) = h_i(f_1(x, \overrightarrow{s}), ..., f_k(x, \overrightarrow{s}), x, \overrightarrow{s})$. *Then* $[[f_i]](t, \overrightarrow{s}) =$
  - $(\bot, w)$ *if* $t = (\bot, w)$.
  - $w + [[g_i]](\overrightarrow{s})$ *if* $t = (0, w)$
  - $w + [[h_i]]([[f_1]](r, \overrightarrow{s}), ..., [[f_k]](r, \overrightarrow{s}), r, \overrightarrow{s})$ *if* $t = (S, w)\ r$
- *If* f *is defined by alternate recursion and (for simplicity of notations I assume* $k = 2$*) the recursive equations are :* $f(0, y, \overrightarrow{s}) = g_1(y, \overrightarrow{s})$, $f(Sx, 0, \overrightarrow{s}) = g_2(x, \overrightarrow{s})$ *and* $f(Sx, Sy, \overrightarrow{s}) = h(f(x, y, \overrightarrow{s}), x, y, \overrightarrow{s})$. *Then* $[[f]](t_1, t_2, \overrightarrow{s}) =$
  - $(\bot, w_1)$ *if* $t_1 = (\bot, w_1)$.
  - $w_1 + [[g_1]](t_2, \overrightarrow{s})$ *if* $t_1 = (0, w_1)$
  - $(\bot, w_1 + w_2)$ *if* $t_1 = (S, w_1)\ r$ *and* $t_2 = (\bot, w_2)$.
  - $w_1 + w_2 + [[g_2]]((r_1, \overrightarrow{s})$ *if* $t_1 = (S, w_1)\ r_1$ *and* $t_2 = (0, w_2)$.
  - $w_1 + w_2 + [[h]](([[f]](r_1, r_2, \overrightarrow{s}), r_1, r_2, \overrightarrow{s})$ *if* $t_1 = (S, w_1)\ r_1$ *and* $t_2 = (S, w_2)\ r_2$.

Comments and examples

1.
$$\begin{array}{lcl} [[add]](S(0)[x], S^\omega[y]) & = & (S, x_0)\ (S, x_1y_0)\ (S, y_1)\ (S, y_2) \cdots \\ [[add]](S(0)[x], S^2(\bot)[y]) & = & (S, x_0)\ (S, x_1y_0)\ (S, y_1)\ (\bot, y_2) \\ [[add]](S^2(\bot)[y], S(0)[x]) & = & (S, y_0)\ (S, y_1)\ (\bot, y_2) \end{array}$$

2. Since this paper is only concerned with decidability results (and not with complexity results), I do not care on the strategy of reduction used to transform the equations into algorithms. However, the strategy that is implicit in this definition is *call by name* : intuitively, at each step the leftmost outermost redex is reduced and, in particular, two copies of the same redex will be reduced twice if they are needed twice.

3 The main result

In this section, I give the main result (theorem 14) of the paper and the definitions that are necessary for its statement.

Definition 10 *Let* $t = (e, \lambda)$ *be a trace.*

1. *The branch of* t *(denoted by* $Br(t)$*) is the word defined by:* $Br(t) = \sum_{k \in Acc(e)} \lambda(k)$.
2. *A letter* x *is unbounded (respectively bounded) in* t *if* $\{j \ /\ x_j \text{ occurs in } Br(t)\}$ *is infinite (respectively finite).*
3. t *is ultimately obstinate if it has at most one unbounded letter.*
4. $Nb(t, x, n)$ *is the least* k *such that* x_n *occurs in* $\lambda(k)$. *If* x_n *does not occur in* $Br(t)$, $Nb(t, x, n)$ *is undefined.*

Comments and examples

- The intuitive meaning of ultimate obstination is that, if the trace represents an infinite computation, *at most one argument* may be used entirely and thus the computation cannot shift from one argument to another one.

- Let $t = e[x]$ be a named element. Since x is the only letter that appears in $Br(t)$, t is ultimately obstinate. Since both x and y are unbounded in $t' = (\perp, \sum_{k\geq 0} x_k\ y_k)$, t' is not ultimately obstinate.

- If t represents a computation, $Nb(t, x, n)$ represents the number of output symbols *produced before the use* of the cell n of the argument x.

- Let $t = (S, x_0)(S, x_1y_0)(0, x_2y_1)$. $Nb(t, x, 0) = 0$, $Nb(t, y, 0) = 1$ and $Nb(t, y, 1) = 2$.

- Let f be a *prc* and assume that, for some function g (usually called the *intentional* behaviour of f), $f(S^n(\perp)) = S^{g(n)}(\perp)$. We will see (cf. proposition 28) that, if $t = [[f]](S^\omega(x))$ and g is not eventually constant, then $g(n) = Nb(t, x, n)$.

Definition 11 *Let C be a class of functions from N into N. Say that C is closed by :*

- finite change *if $f \in C$ and for all n, except finitely many, $f(n) = g(n)$ then $g \in C$.*

- minimum *if $f, g \in C$ then $h \in C$, where h is defined by $h(n) = min\{f(n), g(n)\}$.*

- iteration *if $f \in C$ is such that $f(n) > n$ for all n and g satisfies $g(n+1) = f(g(n))$ for all n, then $g \in C$.*

- multi-step iteration *if $f \in C$ is such that $f(n) > n$ for all n and, for some $p \geq 1$, g satisfies $g(n+p) = f(g(n))$ for all n, then $g \in C$.*

- mixed iteration *if $f, g \in C$, $f(n) > n$ for all n and h satisfies $h(n+1) = Min\{g(n+1), f \circ h(n)\}$ for all n, then $h \in C$.*

Definition 12 *1. Let C_0 be the following set of functions : $\{n \longmapsto 0, n \longmapsto n, n \longmapsto n+1, n \longmapsto$ if $n = 0$ then 0 else $(n-1)\}$.*

2. Let C_{pr} be the least set of increasing *functions containing C_0 and closed by composition, finite change and iteration.*

3. Let C_{mut} be the least set of increasing *functions containing C_0 and closed by composition, finite change and multi-step iteration.*

4. Let C_{alt} be the least set of increasing *functions containing C_0 and closed by composition, finite change, minimum and mixed iteration.*

Open question

I believe that $C_{pr} = C_{alt}$ i.e. C_{pr} is closed by minimum and mixed iteration, but I have not been able to prove that.

Definition 13 *Let C be either C_{pr} or C_{mut} or C_{alt}.*

1. $T(C)$ is the set of traces t such that for every letter x which is unbounded in t, the function $n \longmapsto Nb(t, x, n)$ is in C.

2. The description of t (denoted by $Desc(t)$) is, for $t \in T(C)$, the following set of informations :

- $Val(t)$.
- *for every letter x, whether x is bounded or not in t and*
  - *if x is bounded, $max\{j \ / x_j$ occurs in $Br(t)\}$.*
  - *if x is unbounded, a description of the function $n \longmapsto Nb(t,x,n)$ as a member of C.*

Theorem 14 *1. Let $f \in prc$ and assume t_1, ..., $t_n \in T(C_{pr})$ are ultimately obstinate. Then $[[f]](t_1, ..., t_n) \in T(C_{pr})$ and is ultimately obstinate.*

2. Let $f \in prc_{mut}$ and assume t_1, ..., $t_n \in T(C_{mut})$ are ultimately obstinate. Then $[[f]](t_1, ..., t_n) \in T(C_{mut})$ and is ultimately obstinate.

3. Let $f \in prc_{alt}$ and assume t_1, ..., $t_n \in T(C_{alt})$. Then $[[f]](t_1, ..., t_n) \in T(C_{alt})$.

Moreover, in all cases, $Desc([[f]](t_1, ..., t_n))$ can be computed from $f, Desc(t_1)$, ..., $Desc(t_n)$.

As a consequence, I get :

Corollary 15 *1. If t_1, ..., t_n are named elements and $f \in prc$ (resp. prc_{mut}, prc_{alt}), then $[[f]](t_1, ..., t_n) \in T(C_{pr})$ (resp. $T(C_{mut})$, $T(C_{alt})$).*

2. The following problem is decidable.

Data : Let $t = [[f]](t_1, ..., t_n)$ where $f \in prc$ (resp. prc_{mut}, resp. prc_{alt}) and t_1, ..., t_n are named elements .

Question : What is $Val(t)$? Is the letter x unbounded in t ? If it is bounded what is the maximum n such that x_n occurs in $Lab(t)$? Otherwise what is the function $n \mapsto Nb(t,x,n)$?

3. There is no $f \in prc$ (resp. prc_{mut}, prc_{alt}) such that, for every n, $f(S^n(\perp)) = S^{n^2}(\perp)$.

Proof. 1. and 2. are immediate consequences of theorem 14. It follows easily from proposition 18 below that the function $n \mapsto n^2$ is not in C_{pr} neither in C_{mut} nor in C_{alt}. The point 3 follows thus from proposition 28 below. ■

Comments and open questions

1. It is already proved in [9] that if $f \in prc$ (resp. prc_{mut}) and t_1, ..., t_n are ultimately obstinate then so is $[[f]](t_1, ..., t_n)$. The new result is the *constructivity*. Some informations about $t = [[f]](t_1, ..., t_n)$ can be obtained by a simple computation. For example, *if we know that $Val(t) \geq S(\perp)$*, it is easy to compute $Lab(t)(0)$. But deciding whether $Val(t) \geq S(\perp)$ or not is not immediate at all.

2. It is not difficult to check that, for every function $f \in C_{pr}$ (resp. C_{mut}, C_{alt}), there is a $g \in prc$ (resp. prc_{mut}, prc_{alt}) such that for every n, $g(S^n(\perp)) = S^{f(n)}(\perp)$.

3. It is clear that the ultimate obstination theorem does not hold for prc_{alt} and thus the computation of $Desc([[f]](t_1, ..., t_n))$ has nothing to do with the ultimate obstination ! In particular, parts (1) and (2) of theorem 14 could be stated without the hypothesis on ultimate obstination but, in this case, in the definition of the class C, I should assume that C is closed by minimum and mixed-iteration.

4. Let $prc_{mut,k}$ be the set defined as prc_{mut} but where at most k functions may be defined by simultaneous recursion. Let $C_{mut,k}$ be the set defined as C_{mut} but where the multi-step iteration is restricted to $1 \leq p \leq k$. It follows immediately from the proof that the theorem holds for $prc_{mut,k}$ and $T(C_{mut,k})$.

   Is there $f \in prc_{mut,2}$ such that, for every n, $f(S^n(\perp)) = S^{[n/3]}(\perp)$ where $[x]$ is the integer part of x ? I believe there is no such f but I have not been able neither to prove nor to disprove it. See the remark after proposition 18.

5. Various necessary conditions are known (theorem 14 gives one and some others are given in section 5) for $t \in T$ to be, for example, $[[f]](S^\omega[x], S^\omega[y])$ for some $f \in prc$ but these conditions are far from being sufficient. Can we find other necessary such conditions ? In other words, can we find other properties of traces preserved by the use of prc ?

4 A combinatorial result

The main difficulty in the proof of theorem 14 (which is done by induction on f) is the computation of $Val(t)$ for $t = [[f]](S^\omega(x))$ when f is defined by recursion (for simplicity I assume here that f has only one argument). For the usual case of primitive recursion (the other ones are conceptually the same but technically more difficult) the rough idea of this computation is the following :

For some h and by definition, $t = x_0 + [[h]]([[f]](s), s)$ where $S^\omega(x) = \langle (S, x_0)\ s \rangle$. Let $\tau = x_0 + [[h]](S^\omega[y], s)$ where y is a fresh letter. By the induction hypothesis, I can compute $Desc(\tau)$. I will show that if $Nb(\tau, y, n) \leq n$ for some n then $Val(t) = S^n(\perp)$ where n is the least such integer and otherwise $Val(t) = Val(\tau)$.

By the induction hypothesis, the function $n \longmapsto Nb(\tau, y, n)$ is in C_{pr}. I thus have to show that for functions in C_{pr}, I can effectively decide whether there is an n such that $Nb(\tau, y, n) \leq n$ or not.

The main ingredient of the proof is thus :

Proposition 16 *The following problem is decidable : given a description of f in C_{pr} (resp. C_{mut}, resp. C_{alt}) is there an n such that $f(n) \leq n$?*

Proof. This follows easily from proposition 18 below. ■

Definition 17 *Let f be a function from N to N.*

1. *f is linear if, for n large enough, $f(n) = an + b$ where $a \in N$ and $b \in Z$.*
2. *f is quasi-linear if, for n large enough, $f(n + q) = f(n) + p$ where $p, q \in N$.*
3. *f is N-exponential (resp. Q-exponential) if, for n large enough, $f(n) \geq a^n b$ where b is a positive rational number and $a \in N - \{0, 1\}$ (resp. $a \in Q$).*

Remark

It is easy to check that f is quasi-linear iff there are integers p, q and a function g such that, for all n, $f(n) = [np/q] + g(rm(n, q))$ where $[x]$ denote the integer part of x and $rm(n, q)$ the remainder of n in the division by q.

Proposition 18 *Let C be either C_{pr} or C_{mut} or C_{alt} and let $f \in C$. Then, for n large enough :*

1. *If $C = C_{pr}$ or C_{alt} : f is constant or strictly increasing.*
2. - *If $C = C_{pr}$ or C_{alt} : f is linear or N-exponential.*

- *If $C = C_{mut}$: f is quasi-linear or Q-exponential.*

3. *Moreover, this is effective i.e. we can compute, from a description of f, the various numbers involved.*

The proof is given in the appendix : for *pr* and *alt* this is, more or less, a straightforward verification by case analysis but for *mut* this is a highly non trivial result.

Remark

Let E be a set of integers. Say that f is E-quasi-linear if, for some $p \in E$ and some q, $f(n+p) = f(n) + q$ for all n. If I could prove that each $f \in T(C_{mut,2})$ is either exponential or E-quasi-linear for some E such that $3 \notin E$, I will be able to prove that there is no f such that $f(S^n(\perp)) = S^{[n/3]}(\perp)$. The problem is the following : since, if f and g are $\{p\}$-quasi-linear, $f \circ g$ is $\{p^2\}$-quasi-linear, E will contain $\{2^n \ / \ n \in N\}$. The proof of proposition 18 shows that E will also contain the length of the cycles associated to the functions that are iterated. It is not difficult to find an example of lemma 43 where $p = 4$ and there is a cycle of length 3. This example does not give the desired counter-example but it seems to show that a finer analysis is necessary.

5 Some basic facts on traces

This section recalls various properties of traces. Most of the results are given here without proof : complete proofs, for *prc*, are given in [9]. It is easy to check that they remain valid for prc_{mut} and prc_{alt}. It also introduces the crucial notion of deficiency. The important points are the following.

1. To compute $[[f]](t)$ it is enough to compute $[[f]](e[x])$ where x is a fresh letter and $e = Val(t)$ and then substitute in the result each x_n by $\lambda(n)$ where λ is the labelling function of t. See definition 29 and theorem 30.

2. In a computation, a cell may not be accessed before the previous cells have been accessed. See definition 21 and proposition 22.

3. Let e, e' be lazy integers. If a cell is not used in the computation of $f(e)$ and e, e' coincide up to this cell, then $[[f]](e[x]) = [[f]](e'[x])$. See proposition 25.

4. Assume f is defined by recursion, $r = S^\omega(x)$ and $t = [[f]](r)$. Then, $t = x_0 + [[h]]([[f]](s), s)$ where $s = r\langle x+1\rangle$ (i.e. r with a lift of the indices) and thus t satisfies the equation $t = v[y := t\langle x+1\rangle]$ where $v = x_0 + [[h]](e[y], s)$ and $e = Val(t)$. The important point to compute e is whether or not y is deficient in v i.e. v needs more information on y than it has already produced.

5. Propositions 31 and 33 give the behaviour of Nb and $Desc$ with respect to substitutions.

6. Proposition 28 is the technical result that proves point 3 of corollary 15.

5.1 Deficiency

Definition 19 *Let t be a trace and x be a letter. I say that x is deficient in t if for some n, $Nb(t, x, n) \leq n$.*

Examples and comment

1. Let $t = (S, x_0)(S, x_1y_0)(0, x_2y_1)$. Then, x is deficient in t because $Nb(t, x, 0) = 0$ but y is not because $Nb(t, y, 0) = 1$ and $Nb(t, y, 1) = 2$.

2. The intuition is the following : if t represents a computation, x is deficient in t if, for some n, the n-th cell of x has been used to compute $Val(t)(0), ..., Val(t)(n)$.

3. We will see (cf. the comments after proposition 22) that, for the traces t we have to consider, the function $n \longmapsto Nb(t, x, n)$ is increasing and x is deficient in t iff, for some n, $Nb(t, x, n) = n$.

5.2 Finiteness

Proposition 20 *Let f be a prc and $t_1, ..., t_n$ be finite traces. Then $[[f]](t_1, ..., t_n)$ also is finite.*

5.3 Regularity

The regularity intuitively means that, in a computation, a cell may not be accessed before the previous cells have been accessed. Regularity is called safety in [2].

Definition 21 *Let t be a trace. A letter x is regular in t if for all $n \leq n'$ such that $x_{n'}$ occurs in $Br(t)$, x_n also occurs in $Br(t)$ and the first occurrence of x_n is earlier than the first occurrence of $x_{n'}$.*

Proposition 22 *Let f be a prc and $t_1, ..., t_n$ be traces. Assume that a letter x is regular in each of the t_i, then x also is regular in $[[f]](t_1, ..., t_n)$.*

Comment and examples

- x is regular in $e[x]$ for every element e. x is not regular neither in $(\perp, x_1\ x_0)$ nor in $(\perp, x_0\ x_2)$.
- Assume that x is regular in t. It is clear then that the function $Nb(t, x, n)$ is defined on an initial segment of N and is increasing. Since it is impossible that $Nb(t, x, n) < n$ for all n, x is deficient in t iff, for some n, $Nb(t, x, n) = n$.

5.4 Restrictions

Definition 23 *1. Let w be a word. $w \downarrow x_n = w$ if x_n does not occur in w and otherwise $w' + x_n$ where w' is the longest prefix of w that does not contain an occurrence of x_n.*

2. Let t be a trace. $t \downarrow x_n$ is defined by :

- $(\perp, w) \downarrow x_n = (\perp, w \downarrow x_n)$.
- $(0, w) \downarrow x_n = (\perp, w \downarrow x_n)$ *if x_n occurs in w and otherwise* $(0, w)$.
- $\langle (S, w)\ t \rangle \downarrow x_n = (\perp, w \downarrow x_n)$ *if x_n occurs in w and otherwise* $\langle (S, w)\ t \downarrow x_n \rangle$.

Comment and examples

1. $w \downarrow x_n$ is the word obtained by truncating w *after* the first occurrence (if any) of x_n. $t \downarrow x_n$ is the trace obtained by truncating t at the first node where x_n occurs.
2. $S^\omega[x] \downarrow x_n = S^n(\perp)[x]$.

Proposition 24 *Let f be a prc, x be a letter, $k \in N$ and $t_1, ..., t_n$ be traces. Then $[[f]](t_1, ..., t_n) \downarrow x_k = [[f]](t_1 \downarrow x_k, ..., t_n \downarrow x_k)$*

Proposition 25 *Let f be a prc, $r = e[x], s = e'[x]$ and $\overrightarrow{t}$ be a sequence of elements of $\mathcal{N}$ with names distinct from x. Assume j is accessible both in e and e'. Then*

1. $[[f]](r, \overrightarrow{t}) \downarrow x_j = [[f]](s, \overrightarrow{t}) \downarrow x_j$.
2. *Assume x_j does not occur in $Br([[f]](r, \overrightarrow{t}))$. Then $[[f]](r, \overrightarrow{t}) = [[f]](s, \overrightarrow{t})$*

5.5 Compatibility

Definition 26 *Let s, t be traces. A letter x is compatible with s in t if x is regular in t and, for each n,*

1. *If x_n occurs in $Br(t)$ then $n \in Acc(s)$.*
2. *If $Val(s)(n) = \perp$, then $t = t \downarrow x_n$.*

Comment and examples

1. The intuition for the clause (2) in the definition is the following : if a cell n is filled with $\perp$ in s (this means a lack of information) and the information of this cell is needed in a computation (this means that x_n occurs), then the computation has to stop.
2. x is compatible with s intuitively means that t can be seen as a computation using an argument $e[x]$ where $e = Val(s)$ i.e. the occurrences of x in t are compatible with the value of t.
3. Let e be an element, λ_1 and λ_2 be labelling functions for e. A letter x is compatible with (e, λ_1) in t iff x is compatible with (e, λ_2) in t.
4. Let $s = (e, \lambda)$. Then x is compatible with s in $e[x]$. Let $t = (\perp, \sum_{k \geq 0} x_k)$. Then x compatible with (S^ω, λ) in t but, because clause (1) is not satisfied, x is not compatible with $(S^n(0), \lambda)$ in t for any n. Because clause (2) is not satisfied, x is not compatible with $(S(\perp), \lambda)$ in $(\perp, x_0 x_1 x_0)$.

Proposition 27 *Let f be a prc.*

1. *Assume x is compatible with s in each of the t_i. Then x is compatible with s in $[[f]](t_1, ..., t_n)$.*
2. *For each i, x^i is compatible with $e_i[x^i]$ in $[[f]](e_1[x^1], ..., e_n[x^n])$.*

Proposition 28 *Let f be a prc. Assume that, for each n, $f(S^n(\perp)) = S^{g(n)}(\perp)$ and g is not eventually constant. Then, for each n, $g(n) = Nb(t, x, n)$ where $t = [[f]](S^\omega(x))$.*

Proof. x is unbounded in t because, otherwise, by proposition 25, $Val(t_n)$ would be eventually constant where $t_n = [[f]](S^n(\perp)(x))$. By proposition 25, $t \downarrow x_n = t_n \downarrow x_n$. By proposition 27 and definition 26, clause (2) $t_n = t_n \downarrow x_n$ and thus $t \downarrow x_n = t_n$. The result follows then from the definition of $Nb(t, x, n)$. ■

5.6 Substitutions

The notion of composition is crucial when functions are studied but, usually, only the *results* are, in some sense, composed. The notion of traces allows to compose also the *computations*. The precise meaning of this is given in theorem 30. It needs the notion of substitutions.

Definition 29 *Let t be a trace, $(s_i) = (e_i, \lambda_i)$ be a sequence of traces and (x^i) be a sequence of distinct letters. Assume that, for each i, x^i is compatible with s_i in t. Then $t[x^i := s_i / i = 1, ..., n]$ is the trace obtained by simultaneously replacing each $(x^i)_n$ by $\lambda_i(n)$ in all the words $\lambda(m)$ for $m \in Acc(t)$.*

Example

Let $t = (e, \lambda)$. Then $t = e[x][x := t]$.

Theorem 30 *Let f be a prc, $t_1, ..., t_n$ be traces and, for each i, let r_i be the named element (with the fresh name x^i) such that $Val(t_i) = Val(r_i)$. Then $[[f]](t_1, ..., t_n) = [[f]](r_1, ..., r_n)[x^i := t_i \ / \ i = 1, ..., n]$.*

Proposition 31 *Let t, r be traces and x, y be letters. Assume that x is compatible with r in t and let $s = t[x := r]$.*

Then $Nb(s, y, n) = Min\{Nb(t, y, n), Nb(t, x, Nb(r, y, n)\}$.

Proof. If the least occurrence of y_n in $Br(s)$ comes from t, then $Nb(s, y, n) = Nb(t, y, n)$. Otherwise it comes from the substitution of x_m by $Lab(r)(m)$ where $m = Nb(r, y, n)$ and thus $Nb(s, y, n) = Nb(t, x, Nb(r, y, n)\}$. ■

Corollary 32 *Let $t, r \in T(C_{alt})$, x a letter compatible with r in t. Then $s = t[x := r] \in T(C_{alt})$ and we can compute $Desc(s)$ from $Desc(t)$ and $Desc(r)$.*

Proof. This follows immediately from proposition 31. ■

Proposition 33 *Let $t, r \in T(C_{pr})$ (resp. $T(C_{mut})$), x a letter compatible with r in t. Assume that r, t are ultimately obstinate, then $s = t[x := r] \in T(C_{pr})$ (resp. $T(C_{mut})$) and we can compute $Desc(s)$ from $Desc(t)$ and $Desc(r)$.*

Proof. First note that, if it is true that C_{pr} and C_{mut} are closed by minimum (see the open question in section 3) , the result would follow immediately from proposition 31.

The informations concerning bounded letters are easily computed. Assume y is unbounded in s and let $r = (e, \lambda)$.

1. Assume x is bounded in t and let n be the maximum index of x in $Br(t)$.
   (a) If $e(n) \neq \perp$ then $\lambda(n)$ is finite and, then for p large enough (and easily computed) $Nb(s, y, p) = Nb(t, y, p)$.
   (b) If $e(n) = \perp$ then, by definition 26, $t = t \downarrow x_n$ and $\lambda(n)$ is a final segment of $Br(s)$. Thus, for p large enough, $Nb(s, y, p) = n$.
2. Assume x is unbounded in t. Since t is ultimately obstinate, y is bounded in t. It follows immediately that y must be unbounded in r and then it is easy to check that for n large enough $Nb(s, y, n) = Nb(t, x, Nb(r, y, n))$. ■

6 Proof of theorem 14(1)

The proof is by induction on the definition of f. The only non trivial case is when f is defined by recursion. By theorem 30 and proposition 33, I may assume that t_1, ..., t_n are named elements. Let $\overrightarrow{r} = t_2, ..., t_n$.
(*Case 1 : t_1 is finite*) I only consider $t_1 = S^k(0)[x]$. The proof is similar for $t_1 = S^k(\bot)[x]$. This is done by induction on k.

- $k = 0$: trivial.

- $k = p + 1$: let $t = [[f]](t_1, \overrightarrow{r}) = x_0 + [[h]]([[f]](s, \overrightarrow{r}), s, \overrightarrow{r})$ where $t_1 = \langle (S, x_0)\ s \rangle$. Let $\sigma = S^p(0)[z]$ where z is a fresh variable and $t' = [[h]]([[f]](\sigma, \overrightarrow{r}), \sigma, \overrightarrow{r})$. By the induction hypothesis, $Desc(t')$ is computable. It is easy to check that, for every letter y in $\overrightarrow{r}$, $Nb(t, y, n) = Nb(t', y, n)$ and that, for $n \geq 1, Nb(t, x, n) = Nb(t', z, n-1)$.

(*Case 2 : t_1 is infinite*) The idea is the following : let $t_1 = S^\omega[x] = \langle (S, x_0)\ s \rangle$ and $t = [[f]](t_1, \overrightarrow{r}) = x_0 + [[h]]([[f]](s, \overrightarrow{r}), s, \overrightarrow{r})$. Let y be a fresh letter and $\tau = x_0 + [[h]](S^\omega[y], s, \overrightarrow{r})$. By the induction hypothesis, $\tau \in T(C_{pr})$ and we can compute $Desc(\tau)$. I will show (this is point (1) in each of the sub-cases below) that, if y is not deficient in τ then $Val(t) = Val(\tau)$ and, otherwise, $Val(t) = S^n(\bot)$ where n is the least such that $Nb(\tau, y, n) = n$. The computation of $Desc(t)$ and the proof that $Desc(t) \in T(C_{pr})$ is quite simple (this is point (2) in each of the sub-cases below).

Let $e = Val(t)$ and $v = x_0 + [[h]](e[y], s, \overrightarrow{r})$. Since $s = t_1\langle x + 1 \rangle$, t satisfies the equation $t = v[y := t\langle x + 1 \rangle]$. It is thus not difficult to prove that $t = Lim(v_i)$ where v_i is defined by $v_0 = v$ and $v_{i+1} = v_i[y := v\langle x + i + 1 \rangle]$. For a complete proof see [9]. Note that $Val(t) = Val(v) = Val(v_i)$ for each i. I need first the following result.

Claim 34 *Assume y_m occurs in τ and y is not deficient in $\tau \downarrow y_m$. Then $Val(v) \geq S^{m+1}(\bot)$.*

Proof. Note that the hypothesis means that, for every $n < m$, y_n does not occur in $\sum_{k \leq n} \lambda(k)$. The proof is by induction on m.

$m = 0$: by proposition 25, $v \downarrow y_0 = \tau \downarrow y_0$. By the hypothesis $Nb(\tau, y, 0) > 0$ and so $Val(v) \geq S(\bot)$.

$m = p+1$: by the induction hypothesis, $Val(v) \geq S^m(\bot)$. Thus (by proposition 25) $v \downarrow y_m = \tau \downarrow y_m$. By the hypothesis $Nb(\tau, y, m) > m$ and so $Val(v) \geq S^{m+1}(\bot)$.

■

(*Case 2.a*) Assume first that y is deficient in τ and let n be the least such that $Nb(\tau, y, n) = n$. Note that, since the function $n \longmapsto Nb(\tau, y, n)$ belongs to C_{pr}, we can decide whether y is deficient in τ and, in this case, determine n. Let $\lambda_i = Lab(v_i)$. It is easy to check (by induction on i) that for each i , $Nb(v_i, y, n) = n$.

1. By claim 34, $Val(t) \geq S^n(\bot)$. Since $t = Lim(v_i)$, both $Val(t) = S^n(0)$ and $Val(t) \geq S^{n+1}(\bot)$ are impossible : this would contradict the fact y_n occurs in $\lambda_i(n)$. Thus $Val(t) = S^n(\bot)$.

2. For a letter z in $\overrightarrow{r}$ it is easy to see that the maximum index of z in t is the one it has in $v \downarrow y_n = \tau \downarrow y_n$. The only variable which is unbounded in t is x and $Nb(t, x, p) = n$ for p large enough.

(*Case 2.b*) Assume next that y is bounded and not deficient in τ.

1. Let n be the greatest such that y_n occurs in τ. Since $Desc(\tau)$ can be computed, this n can be determined. By claim 34, $Val(t) \geq S^{n+1}(\bot)$. But then, by proposition 25, $v = \tau$ and thus $Val(t) = Val(v) = Val(\tau)$.

2. t is obtained from v by finitely many substitutions and the result follows from proposition 33.

(*Case 2.c*) Assume finally that y is unbounded and not deficient in τ.

1. Since y is not deficient, $Val(\tau) = S^\omega$. By claim 34, $Val(t) \geq S^n(\bot)$ for every n and thus $Val(t) = S^\omega$.

2. Since $Val(t) = S^\omega$, $v = \tau$. Again, since $t = Lim(v_i)$, it is clear that the only unbounded letter in t is x and the maximum index of the other variables is the one they have in v. Since we know that v is ultimately obstinate, x is bounded in v. Let k be the maximum index of x in v. Since $t = v[y := t\langle x+1\rangle]$, by proposition 31, $Nb(t, x, n+1) = Min\{Nb(v, x, n+1), Nb(v, y, Nb(t\langle x+1\rangle, x, n+1))\}$. It is clear that $Nb(t\langle x+1\rangle, x, n+1) = Nb(t, x, n)$. For $n > k$, x_{n+1} does not occur in $Br(v)$ and thus the previous equation becomes $Nb(t, x, n+1) = Nb(v, y, Nb(t, x, n))$. Since y is not deficient in v, $Nb(v, y, j) > j$ for all j, and thus this equation finishes the proof since it shows that the function $n \longmapsto Nb(t, x, n)$ belongs to C_{pr}.

7 Proof of theorem 14(2)

By induction on f. I only check the difficult case where $t_1 = S^\omega[x]$ and f is defined by mutual recursion.

Assume thus that $f_1, ..., f_k$ are defined by mutual recursion by : $f_i(Sn, \overrightarrow{r}) = h_i(f_1(n, \overrightarrow{r}), ..., f_k(n, \overrightarrow{r}), n, \overrightarrow{r},)$. Let $e_i = [[f_i]](S^\omega[x], \overrightarrow{r})$ and $a_i = Val(e_i)$. I must show that, if $\overrightarrow{r}$ are named elements, the e_i are in $T(C_{mut})$ and I can compute $Desc(e_i)$.

The e_i satisfy the equations $e_i = x_0 + [[h_i]](e_1\langle x+1\rangle, ..., e_k\langle x+1\rangle, \sigma, \overrightarrow{r})$ where $t_1 = \langle (S, x_0)\ \sigma\rangle$ (recall that $\sigma = t_1\langle x+1\rangle$). In the following the $e_i\langle x+1\rangle$ will be called "the recursive calls".

7.1 Computation of a_i : introduction

The algorithm given in the next section computes the a_i. It is essentially the same as in the proof of theorem 14(1), though the mutual recursive calls make the detection of deficiency (corresponding here to loops) harder. If this algorithm had to be implemented, many points could be done in a more efficient way. I chose not to do so because it would be more difficult to understand how it works.

- We keep the informations we already have in two sets : `known_results` and `known_facts`. $a_j \in$ `known-results` means that we know $a_j = S^p(0)$, $a_j = S^p(\bot)$ or $a_j = S^\omega$. `known_facts` is a finite set of informations of the form $a_j \geq S^p(\bot)$.

- The expression "$a_j \geq S^p(\bot)$ is known" means that either $a_j \geq S^p(\bot) \in$ `known_facts` or $a_j = S^\omega \in$ `known_results`.

- r_i is what we can compute of e_i using the informations we have at the present time : if $a_j \in$ `known-results` the recursive call $e_j\langle x+1\rangle$ is replaced by $a_j[y(j)]$ and otherwise by $S^\omega[y(j)]$ where $y(j)$ is a fresh name. Each time we have found the real value a_i, we have to recompute the r_i since we started with $a_j = S^\omega$ and we have discovered it is something else. $Lab(r_i)$ will be denoted by λ_i.

- If w is a word, the expression "a_j is sufficiently known for w" means that
  - either $a_j \in$ `known-results`
  - or $y(j)$ is bounded in w and, for all p such that $y(j)_p$ occurs in w, $a_j \geq S^{p+1}(\bot)$ is known.
- The algorithm calls a procedure denoted by `Next`. When `Next(i,n)` is called $a_i \geq S^n(\bot)$ is known but $a_i \geq S^{n+1}(\bot)$ is not known and we try to know more on a_i.
- `Restart` is a Label of the main program. Going to `Restart` means that we have found a new information and we look if there are some other values to compute.
- Loops correspond to deficiency in the proof of theorem 14(1). Two kinds of loops may occur. The first one corresponds to *unbounded* use of recursive calls. It is detected when the following holds for every i such that $a_i \notin$ `known-results` (this is case (3) of the main program) :
  - For every j such that $y(j)$ is bounded in r_i, a_j is sufficiently known for $Br(a_i)$.
  - For some j such that $y(j)$ is unbounded in r_i, a_j is not sufficiently known for $Br(a_i)$. Since r_i is ultimately obstinate this j is unique. In such a case I will say that f_i recursively calls f_j.

  Since there are finitely many simultaneously defined functions there is a loop : f_{i_1} recursively calls f_{i_2} ... that recursively calls f_{i_n} that recursively calls f_{i_1}.
- The second one corresponds to *bounded* use (this is case (3.a) of the procedure `Next`). The set `rec_calls` of pairs (j,p) is used to detect these loops. $(j,p) \in$ `rec_calls` means that we are "inside" the computation of a_j at a point where we know that $a_j \geq S^p(\bot)$ and we try to know more on a_j. A loop is detected when there is a sequence $(i_1,p_1),...,(i_n,p_n)$ such that the computation of $a_{i_1}(p_1)$ needs the computation of $a_{i_2}(p_2)$ that himself needs ... $a_{i_n}(p_n)$ that needs the computation of $a_{i_1}(p_1)$.
- When, in the description of algorithm, I say, for example, "compute $Desc(r_i)$" or "let n be the least such that $g(n) = n$" this can effectively be done : this follows immediately from the induction hypothesis and the properties (see proposition 18) of C_{mut}.

7.2 The algorithm

`Procedure Next (i, n)`
Begin

1. If $Val(r_i) \geq S^{n+1}(\bot)$ and for all j, a_j is sufficiently known for $\sum_{m \leq n} \lambda_i(m)$, then Add $a_i \geq S^{n+1}(\bot)$ to `known_facts` and Goto `Restart`.
2. If $Val(r_i) = S^n(0)$ (resp. $S^n(\bot)$) and for all j, a_j is sufficiently known for $Br(r_i)$, then Add $a_i = S^n(0)$ (resp. $a_i = S^n(\bot)$) to `known_results` and Goto `Restart`.
3. Otherwise, let $y(j)_p$ be the least token in $\sum_{m \leq n} \lambda_i(m)$ such that $a_j \geq S^{p+1}(\bot)$ is not known.

(a) If $(j,p) \in$ `rec_calls`, then Add $a_j = S^p(\bot)$ to `known_results` and Goto `Restart`.

(b) Otherwise, add (i,n) to `rec_calls` and Call `Next (j,p)`.

End (of procedure `Next`).

`Main Program`

Begin

Let `known_facts` $:= \emptyset$ and `known_results` $:= \emptyset$.

`Label : Restart`

Let `rec_calls` $:= \emptyset$.

If every a_j is known then `Exit` else do

For $j = 1, ..., k$ do : let $s(j) = a_j[y(j)]$ if $a_j \in$ `known_results` and $s(j) = S^\omega[y(j)]$ otherwise ; let $r_i = x_0 + [[h_i]](s(1), ..., s(k), S^\omega[x]\langle x+1\rangle, \overrightarrow{r})$; compute $Desc(r_i)$.

1. If there is an i such that $a_i \notin$ `known-results` and for all j, a_j is sufficiently known for $Br(r_i)$: choose such an i, add $a_i = Val(r_i)$ to `known_results` and Goto `Restart`.

2. If there is an i such that $a_i \notin$ `known-results` and j such that $y(j)$ is bounded in r_i and a_j is not sufficiently known for $Br(r_i)$: choose such an i. Let n be maximal such that $a_i \geq S^n(\bot) \in$ `known_facts`. Let $y(j)_p$ be the least token in $\sum_{m \geq n} \lambda_i(m)$ such that $a_j \geq S^{p+1}(\bot)$ is not known. Let `rec_calls` $:= \{(i,n)\}$ and Call `Next` (j,p).

3. Otherwise, a loop is detected. Choose one. To simplify the notations assume the loop is : f_1 recursively calls f_2 that recursively calls ... f_m that recursively calls f_1.

   For $j = 1$ to m, let G_j be the function $n \longmapsto Nb(r_j, y(j+1), n)$ and let $g = G_1 \circ G_2 \circ ... \circ G_m$ where I consider that $y(m+1) = y(1)$.

   - If $g(n) > n$ for every n, then Add $a_1 = S^\omega$ to `known_results` and Goto `Restart.`
   - Otherwise, let n be the least such that $g(n) = n$. Add $a_1 = S^n(\bot)$ to `known_results` and Goto `Restart.`

   End (of `main program`)

7.3 Proof of the algorithm

The proof is essentially the same as the one of theorem 14(1) and uses extensively proposition 25. Two things have to be shown : the informations on the a_i computed by the algorithm are correct and the algorithm terminates.

The following results will be useful.

Claim 35 *Assume that, at some point of the algorithm, $a_i \geq S^{p+1}(\bot)$ is known. Then we can compute $\lambda_i(n)$ for all $n \leq p$.*

Proof. Immediate. ■

Claim 36 *`rec_calls` cannot contain both (i,n) and (i,m) for $n \neq m$.*

Proof. I have to show that the following cannot appear, where I assume, for simplicity of notations, that (i,m) is put in `rec_calls` after two calls of `Next` : the main program set `rec_calls`$=\{(i,n)\}$ and calls `Next`(j,p) which adds (j,p) in `rec_calls` and calls `Next`(i,m) which goes in case (3.b) and adds (i,m) to `rec_calls`.

Assume, towards a contradiction, that this situation does appear. This means that $a_i \geq S^n(\bot)$ is known but $a_i \geq S^{n+1}(\bot)$ is not and $a_j \geq S^p(\bot)$ is known but $a_j \geq S^{p+1}(\bot)$ is not. If `Next`(j,p) calls `Next`(i,m), this means that $a_i \geq S^m(\bot)$ is known but $a_i \geq S^{m+1}(\bot)$ is not and thus $n=m$. Then, `Next`(i,n) does not go in case (3.b) but in case (3.a) since $(i,n) \in$`rec_calls`. Contradiction ■

1. *Proof of correctness.* The fact that each time the algorithm adds an information to `known_facts` or `known_results`, this information is correct is proved as in the proof of theorem 14(1), using proposition 25. The only different case is when a loop is detected : case (3.a) of the procedure Next and case (3) of the main program. In the proof of theorem 14(1), both cases correspond to y deficient in τ. Recall that, in this case, we showed that if n is the least such that $Nb(\tau,y,n)=n$ then $Val(t)=S^n(\bot)$.

   - *The unbounded case.*

     Assume again, for simplicity of notations that the loop is f_1 recursively calls f_2 and f_2 recursively calls f_1. At this point, $y(2)$ (resp. $y(1)$) is the only unbounded letter in r_1 (resp. r_2). Moreover, all the other letters are sufficiently known and, in particular, $y(1)$ (resp. $y(2)$) is sufficiently known in r_1 (resp. r_2). By claim 35, let s_1, s_2 be finite traces such that $e_1 = x_0 + [[h_1]](s_1, e_2\langle x+1\rangle, \sigma, \overrightarrow{r})$ and $e_2 = x_0 + [[h_2]](e_1\langle x+1\rangle, s_2, \sigma, \overrightarrow{r})$ where $\sigma = S^\omega[x]\langle x+1\rangle$.
     Let $\rho_1 = x_0 + [[h_1]](s_1, S^\omega[y(2)], \sigma, \overrightarrow{r})$ and $\rho_2 = x_0 + [[h_2]](S^\omega[y(1)], s_2, \sigma, \overrightarrow{r})$. Let $\tau_1 = \rho_1[y(2) := \rho_2]$ and $\tau_2 = \rho_2[y(1) := \rho_1]$. It is clear that $e_1 = \tau_1[y(1) := e_1\langle x+2\rangle]$ and $e_2 = \tau_2[y(2) := e_2\langle x+2\rangle]$. For more details, see the proof of the ultimate obstination theorem in case of mutual recursion in [9]. For $i=1,2$ let $G_i(n) = Nb(\rho_i, y(i), n)$. It follows from proposition 33 that $Nb(\tau_1, y(1), n) = G_1 \circ G_2(n)$.
     The fact that, if $y(1)$ is not deficient in τ_1 then $a_1 = S^\omega$ and otherwise $a_1 = S^n(\bot)$ where n is the least such that $Nb(\tau_1, y(1), n) = n$ is proved exactly as in the proof of theorem 14(1).

   - *The bounded case.*

     Assume again, for simplicity of notations, that the loop has length 2. By claim 36, assume the loop is $(1,n) \to (2,p) \to (1,n)$. I have to show that $a_1 = S^n(\bot)$. As in the previous unbounded case, I can find τ_1 such that $e_1 = \tau_1[y(1) := e_1\langle x+2\rangle]$ and show that n is the least such that $Nb(\tau_1, y(1), n) = n$. The results follows then exactly as in the proof of theorem 14(1).

2. *Proof of terminaison* :

   - By claim 36, the procedure `Next` can call itself at most k many times.
   - Thus, when the main program calls `Next` the program always return to `Restart` with a new information, i.e. some new $a_i \geq S^n(\bot) \in$ `known_facts` or some new $a_i \in$ `known_results`.
   - In cases 1 and 3 of the main program, the algorithm adds some new a_i in `known_results`. It is thus enough to check that it cannot always stay in case 2. This case corresponds to *bounded* letters : since there is a finite number of traces and letters, there is only a finite set of informations

needed about bounded letters. The result follows then again from the fact that each time the algorithm goes back to `Restart` it has got a new information.

7.4 $e_i \in T(C_{mut})$ and computation of $Desc(e_i)$

Let $r_i = x_0 + [[h_i]](s(1), ..., s(k), \sigma, \overrightarrow{r})$ where $s(j) = a_j[y(j)]$. We know that $e_i = r_i[y(j) := e_j\langle x+1\rangle \ / \ j = 1, ..., k]$.

The only non immediate case is the one when there is a loop (of length p) of recursive calls, e.g. f_1 recursively calls f_2 that recursively calls ... that recursively calls f_1. In this case the only letter with unbounded index in the e_i $(i = 1, ..., p)$ is x. The informations concerning the other variables are easy to get.

Assume again, for simplicity of notations, that the loop has length 2. As in the proof of correctness, I can find traces τ_i such that :

- the only letters occurring in τ_i are : $x, y(i)$ and the letters in $\overrightarrow{r}$.
- the only letter which is unbounded in τ_i is $y(i)$.
- $\tau_i \in T(C_{mut})$ and $Desc(\tau_i)$ can be computed by using proposition 33.
- $e_i = \tau_i[y(i) := \tau_i\langle x+2\rangle]$.
- $y(i)$ is not deficient in τ_i.

It is then not difficult to check (this in done as in the proof of theorem 14(1)) that for n large enough $Nb(e_i, x, n+2) = Nb(\tau_i, y(i), Nb(e_i, x, n))$ and thus the function $n \mapsto Nb(e_i, x, n)$ is in C_{mut}.

8 Proof of theorem 14(3)

The proof is essentially the same as the one of theorem 14(1). The difficult case is when f is defined by alternate recursion. For simplicity of notations, I assume the recursion is made only on two arguments. The main case is : $t = [[f]](S^\omega[x], S^\omega[y], \overrightarrow{r})$. Let $v = x_0 + y_0 + [[h]](Val(t)[z], \sigma, \tau, \overrightarrow{r})$ where $\sigma = S^\omega[x]\langle x+1\rangle$ and $\tau = S^\omega[y]\langle y+1\rangle$ and z is a fresh letter. Let $\rho = x_0 + y_0 + [[h]](S^\omega[z], \sigma, \tau, \overrightarrow{r})$

I show, exactly as in the proof of theorem 14(1), that if z is not deficient in ρ then $Val(t) = Val(\rho)$ and otherwise that $Val(t) = S^n(\perp)$ where n is the least such that $Nb(\rho, z, n) = n$.

It is clear that $t = v[z = t\langle x+1, y+1\rangle]$. It follows easily that in the non trivial case i.e. $Val(t) = S^\omega$:

- If G_1 is the function : $n \mapsto Nb(t, x, n)$ then $G_1(n+1) = Min\{Nb(v, x, n+1), Nb(v, z, G_1(n))\}$. Similarly for $G_2 : n \mapsto Nb(t, y, n)$.
- If G_α is the function : $n \mapsto Nb(t, \alpha, n)$ where α is a letter in $\overrightarrow{r}$, then $G_\alpha(n) = Nb(v, \alpha, n)$.

Thus $t \in T(C_{alt})$. ■

9 The undecidability result

This section uses the general notion of trace where the data type of *lists* is allowed. I do not recall here the corresponding notions. More details can be found in [9].

Theorem 37 *There is no algorithm to compute $f(S^\omega)$ from a description of f, in the following cases :*

1. *f is a prc using integers and lists of integers as data types.*
2. *f is a prc using only integers as data types but allowing the following recursion scheme : $f(Sx, y) = h(f(x, Sy), x, y)$.*

Proof. Note again that these schemata define new algorithms but the functions they compute are primitive recursive functions.

It is enough to show that, from a description of a Turing machine, I can compute a *prc* f such that the Turing machine halts if and only if $f(S^\omega) = S^\omega$.

Assume (without loss of generality) that the final state has number 0 and that when the final state is entered, the machine remains in this state for ever. It is quite usual (and easy) to show that the internal description (the state, the positions of the scanned cells and the symbols in the cells) of the machine at the step n are primitive recursive functions of n. Notice that this definition is usually made by use of mutual recursion but primitive recursive functions are -extensionally- closed by mutual recursion. It is then not difficult to find a *prc state* depending on one argument such that, for every integer n, $state(S^n(0))$ is the number of the state of the machine at time n.

1. Define $incr$ and g by : $incr(nil) = nil$, $incr(cons(a, l)) = cons(Sa, incr(l))$, $g(0) = cons(0, nil)$ and $g(n + 1) = cons(0, incr(g(n)))$. Then $g(S^\omega)$ is the infinite list $[0, 1, 2, ...]$. Define h by: $h(nil) = 0, h(cons(a, l))$ = if $state(a) = 0$ then $Sh(l)$ else $h(l)$ and let $f = h \circ g$. It is easy to check that $f(S^\omega) = S^\omega$ iff the machine enters the final state.
2. Define g and f by $g(0, y) = 0$, $g(Sn, y)$ = if $state(y) = 0$ then $Sg(n, Sy)$ else $g(n, Sy)$. $f(x) = g(x, 0)$. It is again easy to check that $f(S^\omega) = S^\omega$ iff the machine enters the final state. ■

10 Appendix

10.1 Proof of proposition 18 for C_{pr}

By simultaneous induction on the construction of f.

1. For the base functions and finite change, the result is trivial
2. Composition :
   - if f and g are are either constant or strictly increasing, then so is $f \circ g$.
     - if f and g are linear then so is $f \circ g$.
     - if $f(n) = an + b$ and $g(n) \geq d^n c$ then $f(g(n)) \geq d^n ac + b \geq d^n e$ (for some e) for n large enough and $g(f(n)) \geq d^{an+b} c$.
     - if $f(n) \geq ab^n$ and $g(n) \geq d^n c$. Since $d \geq 2$, $g(n) \geq n$ for n large enough and then $f(g(n)) \geq b^n c$ for n large enough .
3. Iteration : since for all n, $f(n) > n$, $g(n + 1) = f \circ g(n) > g(n)$.
   - if $f(n) = cn + d$ and $g(n + 1) = f(g(n))$:
     - c cannot be 0 since f cannot be constant.
     - If $c = 1$, then $d \geq 0$ and $g(n + 1) = g(n) + d$ and so $g(n) = dn + b$.

- If $c > 1$ then $g(n_0+n) = d+c(d+c(d+...))) = d(c^{n+1}-1)/(c-1) \geq c^n e$ (for some e) for n large enough.

- if $f(n) \geq d^n c$ then for some n_0 and all $n > n_0$ $f(n) \geq 2n$ and so $g(n_0+n) \geq 2^n g(n_0)$. ■

10.2 Proof of proposition 18 for C_{mut}

By simultaneous induction on the construction of f.

1. For the base functions and finite change, the result is trivial.

2. Composition : the only problem is to check that the composition of two quasi-linear functions is quasi-linear. Assume $f(n+q) = f(n)+p$ and $f'(n+q') = f'(n)+p'$. Then $f \circ f'(n+qq') = f(f'(n)+qp') = f \circ f'(n)+pp'$.

3. Multi-step iteration : the only difficult case is when f is quasi-linear and h is defined by iteration from f. Assume $f(n+q) = f(n)+p$.

   - if $q > p$: it is easy to check that for some n, $f(n) \leq n$. So this case does not occur.
   - if $q < p$: it is easy to check that for some $a > 1$ and for n large enough $f(n) \geq an$ and thus h is exponential.
   - if $p = q$: this is given by the next lemma. ■

Lemma 38 *Assume that f is increasing, for all n large enough, $f(n+p) = f(n)+p$ and $h(n+r) = f \circ h(n)$. Then h is quasi-linear for n large enough.*

Proof. The idea is the following : since h is obtained by iterating f and the value of $f(n)$ depends essentially of the remainder of n in the division by p, we have to study, for each i, the sequence defined by $a(0,i) = i$ and $a(n+1,i) =$ the remainder of $f(a(n,i))$ in the division by p. Since the number of possible remainders is finite this sequence is eventually cyclic. The main difficulty of the proof is the fact (see lemma 43) that the form of these cycles is essentially independent of i.

I have to show that h is quasi-linear on some final segment of N and to determine this segment. For simplicity, I will assume the hypothesis of the lemma hold for all n. For the general case I should, in the following, replace everywhere "for all n" by "for n large enough" and check that the final segment of N on which the mentioned property is true can be effectively determined. This is easily done and I will not care about this.

The result follows immediately from lemmas 42 and 43 below. I need first some definitions.

Claim 39 $f(0) \leq f(1) \leq ... \leq f(p-1) \leq f(0)+p$.

Proof. This immediately follows from the fact that f is increasing and $f(p) = f(0)+p$. ■

Claim 40 *I may assume without loss of generality, that $0 \leq f(0) < p$.*

Proof. Let $k = [f(0)/p]$ and define f' by : $f'(n) = f(n) - kp$. Clearly, f' is increasing, $0 \leq f'(0) < p$ and, for all n, $f'(n+p) = f'(n)+p$. Define h' by $h'(i) = h(i)$ for $0 \leq i < r$ and $h'(n+r) = f'(h'(n))$. Assume $h'(n+a) = h'(n)+b$ for some a, b. Let $f^{(l)}$denotes $f \circ f ... \circ f$, l times. It is easy to check (by induction on l) that for all $l, i, h(lr+i) = f^{(l)}(h(i)), h'(lr+i) = f'^{(l)}(h'(i))$ and $f^{(l)}(i) = f'^{(l)}(i)+lkp$. It follows that, for all n, $h(n) = h'(n)+[n/r]kp$ and thus $h(n+ar) = h(n)+rb+kap$. ■

Definition 41

- *Let $I = \{i \ / \ 0 \ \leq i < p\}$. For $i \in I$, let $q(i)$ be the quotient and $r(i)$ the remainder, in the division of $f(i)$ by p. Note that q is increasing on i and, by claims 39 and 40, $q(i) = 0$ or 1.*
- *Say that i is a right (resp. left) point if $q(i) = 1$ (resp. $q(i) = 0$). This terminology will be easily understood by looking at the example below.*
- *For $j \in I$, define $a(n,j)$ and $b(n,j)$ by : $a(0,j) = j$, $a(n+1,j) = r(a(n,j))$ and $b(n,j) = q(a(n,j))$. Thus $f(a(n,j)) = b(n,j)p + a(n+1,j)$.*
- *Let $lg(j)$ be the least such that $a(n+lg(j),j) = a(n,j)$, for some n. Say that $a(n,j) \to a(n+1,j) \to ... \to a(n+lg(j)-1,j) \to a(n+lg(j),j) = a(n,j)$ is a cycle for j. The function lg is clearly defined since I is finite. It is also clear that, for $n \geq p$, $a(n+lg(j),j) = a(n,j)$.*
- *Let $S(j) = Card\{m \ / \ n \leq m < n+lg(j) \text{ and } b(m,j) = 1\}$. It is easy to check that $S(j)$ does not depend on n, if $n \geq p$.*

The role of S and the cycles is given by lemma 42 below.

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$f(i)$	4	4	5	5	8	9	9	9	10	13	14	14
$r(i)$	4	4	5	5	8	9	9	9	10	1	2	2
$q(i)$	0	0	0	0	0	0	0	0	0	1	1	1

In this example (where $p = 12$) : $1 \to 4 \to 8 \to 10 \to 2 \to 5 \to 9 \to 1$ is a cycle for 1. $lg(1) = 7$ and $S(1) = 2$. There is, in fact, only one cycle : for example, 6 has the same cycle since $6 \to 9$ and 9 belongs to the cycle of 1. This is however not the general case.

Lemma 42 *For all $n \geq pr$, $h(n+lr) = h(n) + pS$ where $l = lg(i)$, $S = S(i)$ and $i = rh(n-pr)$ is the remainder of $h(n-pr)$ in the division by p.*

Proof. Let $n = pr + n'$, $i = rh(n')$, $l = lg(i)$ and $S = S(i)$. Then $h(n') = pk + i$ for some k. Since $f(a(m,i)) = pb(m,i) + a(m+1,i)$, it is easy to check (by induction on q) that $f^{(q)}(h(n')) = a(q,i) + p(k + \sum_{m=0}^{q-1} b(m,i))$. Since $a(p+l,i) = a(p,i)$ it follows that $f^{(p+l)}(h(n')) = f^{(p)}(h(n')) + p \sum_{m=p}^{p+l-1} b(m,i) = f^{(p)}(h(n')) + pS$. Thus, $h(n' + (p+l)r) = f^{(p+l)}(h(n')) = f^{(p)}(h(n')) + pS = h(n'+pr) + pS$. ■

Lemma 43 *The functions lg and S are constant on I.*

Proof. This is done in several steps, according to various situations. Claim 46 gives the easy cases. Claims 48 to 50 prove the most difficult case. Claims 44 and 45 will be useful.

Claim 44 *There are no n, m and i, j such that $b(n,i) = 0, b(m,j) = 1$ and $a(n+1,i) < a(m+1,j)$.*

Proof. Otherwise, we have $f(0) + p \leq f(a(n,i)) + p = a(n+1,i) + p < a(m+1,j) + p = f(a(m,j)) \leq f(p-1)$ and this contradicts claim 39. ■

Claim 45 *1. Assume $b(n,i) = 1$ and $a(n+1,i) \geq a(n,i)$. Then, for all $m \geq n$, $b(m,i) = 1$. Similarly, if $b(n,i) = 0$ and $a(n+1,i) \leq a(n,i)$ then, for all $m \geq n$, $b(m,i) = 0$.*

2. *Assume that for n large enough $b(n,i) = 1$ (resp. $b(n,i) = 0$). Then, for some m, $a(m+1,i) = a(m,i)$.*

3. *Assume $a(m+1,i) = a(m,i)$ and $b(m,i) = 1$ (resp. $b(m,i) = 0$). Then for each j, there is an n such that $a(n+1,j) = a(n,j)$ and $b(n,j) = 1$ (resp. $b(n,i) = 0$).*

Proof.

1. Since $b(n,i) = 1$ and $a(n+1,i) \geq a(n,i)$, then $b(n+1,i) = 1$. I prove, by induction on m that for all $m \geq n$, $a(m+1,i) \geq a(m,i)$ and $b(m,i) = 1$. Since f is increasing $p + a(m+1,i) = b(m,i)p + a(m+1,i) = f(a(m,i)) \leq f(a(m+1,i)) = b(m+1,i)p + a(m+2,i) = p + a(m+2,i)$ and the result follows.

2. Assume that, for $m \geq n$, $b(m,i) = 1$ and, for example, $a(n+1,i) \geq a(n,i)$. It is easy to prove by induction as in the previous case that for $m \geq n$, $a(m+1,i) \geq a(m,i)$. The result follows immediately from the fact that I is finite.

3. - Assume that for some n_0, $a(n_0,j) \geq a(m,i)$. It is easy to check by induction as in the previous cases that for $n \geq n_0$, $a(n,j) \geq a(m,i)$ and $b(n,j) = 1$. The result follows then from (2).
   - Assume for all n, $a(n+1,j) < a(m+1,i)$. Then, by claim 44, for all n, $b(n,j) = 1$ and again the result follows from (2). ■

Claim 46 *1. One of the following situation holds :*

   (a) All the cycles are uniquely made of right points.

   (b) All the cycles are uniquely made of left points.

   (c) For all i,n if $b(n,i) = 1$ then $b(n+1,i) = 0$.

   (d) For all i,n if $b(n,i) = 0$ then $b(n+1,i) = 1$.

2. In case (a) $lg(i) = S(i) = 1$ for each i. In case (b) $lg(i) = 1$ and $S(i) = 0$ for each i. In case (c) and (d) hold simultaneously $lg(i) = 2$ and $S(i) = 1$ for each i.

Proof.

1. Assume neither case (c) nor case (d) holds. Let $b(n,i) = b(n+1,i) = 0$ and $b(m,j) = b(m+1,j) = 1$. If $a(n,i) \geq a(n+1,i)$ then, claim 45 implies that we are in case (a). Similarly if $a(m+1,j) \geq a(m,j)$ we are in case (b). Otherwise, we have $a(n,i) < a(n+1,i) < a(m+1,j) < a(m,j)$ and this contradicts claim 44.

2. If (a) or (b) holds the result follows immediately from claim 45. Assume (c) and (d) hold simultaneously and, for all n, $a(n+2,i) \neq a(n,i)$. Say, for example, $a(0,i) < a(2,i)$. Since $b(2n,i)$ is constant it is immediate to check that for all n, $a(2(n-1),i) < a(2n,i)$ which is impossible. ■

Thus it remains to prove lemma 43 in the following case (the symmetric one is similar).

- For all i,n if $b(n,i) = 1$ then $b(n+1,i) = 0$.

- For some i, n $b(n,i) = b(n+1,i) = 0$.

Note that the example of definition 41 corresponds to this situation. Fix i such that $b(n,i) = b(n+1,i) = 0$ for some n and let $l = lg(i)$ and $S = S(i)$. Note that $lg(i) \geq 3$ since there are at least two left points and one right point. Choose $j \neq i$. I may assume that $\{a(m,i) \ / \ m \in N\} \cap \{a(m,j) \ / \ m \in N\} = \emptyset$ since, otherwise, i and j have the same cycle and thus, $lg(j) = l$ and $S(j) = S$. I will show that the cycle of j looks like the one of i and thus, again, $lg(j) = l$ and $s(j) = S$. I need some more definitions. I may assume without loss of generality that :

1. $a(l,i) = a(0,i)$ and $a(lg(j),j) = a(0,j)$. This means that, in the cycles corresponding to j (resp. i), the first element of the cycle is j (resp. i).
2. $b(0,i) = b(0,j) = 1$. This means that i and j are right points.
3. For all n, if $b(n,i) = 1$ then $a(n,i) \geq a(0,i)$. This means that i is the smallest of the right points of its cycle. It follows that $a(1,i)$ is the smallest of the left points of its cycle.
4. Let k be such that $b(k,i) = 0$ and for every m such that $b(m,i) = 0$, $a(m,i) \leq a(k,i)$. This means that $a(k,i)$ is the largest of the left points in its cycle. It follows that $a(k+1,i)$ is the largest of the right points in its cycle.
5. There is no m such that $b(m,j) = 1$ and $a(m,j) < a(0,i)$. This means that the smallest of the right points of the cycle of i is smaller than the smallest of the right points of the cycle of j. (The opposite case is done in a similar way). It follows that there is no n such that $b(n,j) = 0$ and $a(n,j) < a(1,i)$.

Definition 47 *1. Say that $Suc(n,m)$ if $a(n,i) < a(m,i)$ and there is no q such that $a(n,i) < a(q,i) < a(m,i)$. It is convenient to also say that $Suc(k+1,1)$.*

2. If $Suc(n,m)$, say $j \in]n,m[$ if $a(n,i) < j < a(m,i)$ (resp. if $n = k+1$, $j > a(k+1,i)$)

Example and comments

1. Note that this notion is only defined modulo l.
2. By the assumption on i and j, we always have $Suc(k,0)$ and $a(m,j) \in]k,0[$ iff $b(m,j) = 0$ and $a(m,\ j) > a(k,i)$.
3. In the example of definition 41, the cycle corresponding to $i = 0$ satisfies $k = 3$ and $a(1,i) < a(5,i) < a(2,i) < a(6,i) < a(3,i) < a(0,i) = a(7,i) < a(4,i)$.

   Thus $Suc(1,5), Suc(5,2), Suc(2,6), Suc(6,3), Suc(3,0), Suc(0,4)$ and $Suc(4,1)$.

Claim 48 *Assume $Suc(n,m)$. Then $Suc(n+1,m+1)$.*

Proof. The non trivial cases (i.e. the ones that do not immediately follow from the fact that f is increasing) are :

- $n = k+1, m = 1$: first note that by claim 44, $a(k+2,i) \leq a(2,i)$. The equality is impossible since this would imply $lg(i) \leq k$ and this contradicts the fact that the cycle has at least $k+1$ points. Assume that $a(k+2,i) < a(n,i) < a(2,i)$ for some n. If $b(n-1,\ i) = 0$ then $a(n-1,i) < a(1,i)$: contradiction. If $b(n-1,i) = 1$ then $a(n-1,i) > a(k+1,i)$: contradiction.
- $b(n,i) = b(m,i) = 0, b(n+1,i) = 0$ and $b(m+1,i) = 1$. I prove below that $]n+1,m+1[=]k,0[$:

- $m+1=l$: otherwise $a(m+1,i) > a(l,i)$ and thus $a(n,i) < a(l-1,i) < a(m,i)$. This contradicts $Suc(n,m)$.
- $n+1=k$: otherwise $a(n+1,i) < a(k,i)$. If $b(k-1,i)=0$ then $a(n,\ i) < a(k-1,i)$ and since $Suc(n,m)$, $a(k-1,i) \geq a(m,i)$ and thus $b(k,i)=1$. Contradiction. If $b(k-1,i)=1$ then $a(k-1,i)=a(k+1,i)$ and this contradicts the fact that $l \geq 3$. ■

Claim 49 below implies that, for $0 \leq q < q' < l$, $a(q,j) \neq a(q',j)$ and $b(q,j) = b(q,i)$. Thus claim 50 shows that $lg(j)=l$ and $S(j)=S$. This finishes the proofs of lemmas 43, 38 and thus the proof of proposition 18.

Assume that $j \in]n,m[$ for some n,m such that $Suc(n,m)$.

Claim 49 *For all q, $a(q,j) \in]n+q, m+q[$.*

Proof. Note that if the cycle of i satisfies (as in the example of definition 41) $a(1,i) < a(5,i) < a(2,i) < a(6,i) < a(3,i) < a(0,i) = a(7,i) < a(4,i)$ and if $j > a(4,i)$, claim 49 implies $a(1,i) < a(4,j) < a(5,i) < a(1,j) < a(2,i) < a(5,j) < a(6,i) < a(2,j) < a(3,i) < a(6,j) < a(0,i) < a(3,j) < a(4,i) < a(0,j)$ and $a(7,j) > a(4,i)$.

The proof is by induction on q. Using the fact that f is increasing, the only non trivial cases are :

- $]n+q, m+q[=]k+1, 1[$: since $a(q,j) > a(k+1,i)$ we must have $a(q+1,j) > a(k+2,i)$. By claim 44 and the fact that the cycles of i and j are disjoints, we have $a(q+1,j) < a(2,i)$.
- $]n+q+1, m+q+1[=]k, 0[$: $b(q+1,j)=0$ because otherwise, since $a(q,j) < a(m+q,i)$, we should have $a(q+1,j) < a(0,i)$ and this contradicts the assumption on j. Then, since $a(q,j) > a(n+q,i)$, $a(q+1,j) > a(k,i)$ and we are done. ■

Claim 50 $a(l,j)=j$.

Proof. Claim 49 shows that $a(l,j) \in]n,m[$. Assume $a(l,j) \neq j$, e.g. $a(l,j) < j$. It follows from claim 49 by an immediate induction on q that, for every q, $a(q.l,j) \in]n,m[$ and $a(q-1.l,j) < a(q.l,j)$. Contradiction. ■

10.3 Proof of proposition 18 for C_{alt}

By induction on f, as for C_{pr}. The only new case is for f given by mixed iteration. Assume that :

- f, g are strictly increasing and linear or exponential, for n large enough.
- $f(n) > \ n$, for all n.
- $h(n+1) = Min\{g(n+1), f \circ h(n)\}$ (and thus $h(n) \leq g(n)$) for n large enough.

I must prove that :

- h is strictly increasing : $h(n+1) > h(n)$ follows immediately from the facts that $g(n+1) > g(n) \geq h(n)$, $f(h(n)) > h(n)$ and $h(n+1) = Min\{g(n+1), f \circ h(n)\}$.
- h linear or exponential on some final segment of N.

  1. Assume that for for $n \geq m$, $g(n) = an + b$ and $h(n+1) = Min\{g(n+1), ch(n) + d\}$.

(a) Assume first that $c > 1$. There is $n \geq m$ such that $h(n) = g(n)$ and $g(n+1) \leq cg(n) + d$.
Proof : otherwise for all n large enough, $h(n) < g(n)$. Then $h(n+1) = ch(n) + d$. This is impossible since then, for some e, $h(n) \geq c^n e$, for n large enough and this contradicts the assumption $h(n) < g(n)$. *End of proof.*
Then it is easy to see (by simultaneous induction on p) that $h(p) = g(p)$ and $g(p+1) \leq cg(p) + d$ for all $p \geq n$.

(b) Assume that $c = 1$ and $d > a$. There is $n \geq m$ such that $h(n) = g(n)$.
Proof : otherwise for all $n \geq m$, $h(n) < g(n)$. Then $h(n+1) = h(n) + d$ and, for some q, $h(m) \geq dm - q$ for $n \geq m$, and this contradicts the assumption $h(n) < g(n)$. *End of proof.*
Then it is immediate to see (by induction on n) that $h(n) = g(n)$ for all $n \geq m$.

(c) Assume that $c = 1$ and $d \leq a$. If $h(m) + d \leq g(m+1)$, it is immediate to see (by induction on n) that for all $n \geq m$, $h(n+1) = h(n) + d$ and thus h is linear. Otherwise, it is easy to check that $h(m+1) + d \leq g(m+2)$ and the result is similar.

2. Assume that for for $n \geq m$, $g(n) \geq b^n a$ and $h(n+1) = Min\{g(n+1), ch(n) + d\}$.

(a) Assume first that $c > b$. There is $n \geq m$ such that $h(n) \geq b^n a$.
Proof : otherwise for $n \geq m$, $h(n) < b^n a \leq g(n)$. Then, $h(n+1) = ch(n) + d$ and thus $b^n a > h(n) = cn - q$ for some q. This contradicts $c > b$. *End of proof.*
Then, for some $r > 1$ and some e, $h(p) \geq r^p e$ for all $p \geq n$.
Proof : show (by simultaneous induction on p, using $c > b$) that, if $d \geq 0$ then $h(p+1) = ch(p) + d$ and $h(p) \geq b^{p+1} a$ for $p \geq n$ and, otherwise, for $p \geq 0$, $h(n+p) \geq ab^{n+p} + c^{p-1}d + c^{p-2}d + ... + d$. *End of proof.*

(b) Assume $1 < c \leq b$. If for all $n \geq m$, $c.h(n) + d > b^{n+1} a$ then h is exponential. Otherwise let $n \geq m$ be such that $ch(n) + d \leq b^{n+1} a$. It is easy to see (by simultaneous induction on p) that for all $p \geq n$, $ch(p) + d \leq b^{p+1} a$ and $h(p+1) = ch(p) + d$ and thus that h is exponential.

(c) Assume that $c = 1$. If for all $n \geq m$, $h(n) + d > b^{n+1} a$ then h is exponential. Otherwise let $n \geq m$ be such that $h(n) + d \leq b^{n+1} a$. It is easy to see (by simultaneous induction on p) that for all $p \geq n$, $h(p) + d \leq b^{p+1} a$ and $h(p+1) = h(p) + d$ and thus that h is linear.

3. Assume $g(n) \geq b^n a$ and $f(n) \geq d^n c$ for n large enough. Then, for n large enough $f(n) \geq b\, n$. It follows (by induction on n) that for some n_0, $h(n + n_0) \geq Min\{b^{n+n_0} a, b^n h(n_0)\}$ for all n and thus h is exponential.

4. Assume finally $g(n) = an + b$ and $f(n) \geq d^n c$ for n large enough. Let $e = Max\{a, 2\}$. Let m be such that for $n \geq m$, $f(n) > en$ and $g(n) = an + b$. There is $n \geq m$ such that $g(n+1) \leq f \circ h(n)$.

Proof : otherwise for all $n \geq m$, $h(n+1) = f \circ h(n)$ and thus $h(p+m) \geq 2^p h(m)$ which contradicts $eh(n) < f \circ h(n) < g(n+1) = a(n+1) + b$. *End of proof.*

Then it is easy to see (by simultaneous induction on p, using $e \geq a$) that for all $p \geq n$, $g(p+1) \leq f \circ h(p)$ and $h(p) = g(p)$. ■

References

[1] R. Amadio and P.L. Curien. *Domains and Lambda Calculi.* Cambridge University Press Press, 1998.

[2] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265-321, 1982.

[3] L. Colson. About primitive recursive algorithms. *Theoretical Computer Science*, 83 : 57-69, 1991.

[4] L. Colson. Représentation intensionnelle d'algorithmes dans les systèmes fonctionnels. *Thèse de doctorat, Université P 7*, 1991.

[5] L. Colson. A unary representation result for system $\mathcal{T}$. *Annals of Mathematics and Artificial Intelligence*, 16:385-403, 1996.

[6] T. Coquand. Une preuve directe du théorème d'ultime obstination. *Compte Rendus de l'Académie des Sciences*, 314, Serie I, 1992.

[7] R. David. Un algorithme primitif récursif pour la fonction inf. *Compte Rendus de l'Académie des Sciences*, 317 (Série I), 1993.

[8] R. David. The inf function in the system $\mathcal{F}$. *TCS*, 135 : 423-431, 1994.

[9] R. David. On the asymptotic behaviour of primitive recursive algorithms. *TCS*, 266 :159-193, 2001.

[10] Martin Hotzel Escardo. On lazy natural numbers with applications. *SIGACT News*, 24(1), 1993.

[11] D. Fredholm. Intentional aspects of function definitions. *Theoretical Computer Science*, 152 : 1-66, 1995.

[12] D. Fredholm. Computing minimum with primitive recursion over lists. *Theoretical Computer Science*, 163 : 269-276, 1996.

[13] J.-L. Krivine. Un algorithme non typable dans le systeme $\mathcal{F}$. *Compt. Rend. de l'Acad. des Sci. Paris*, 304(5), 1987.

[14] Roza Peter. *Recursive Functions.* Academic Press, 1968.

[15] H. Rogers. *Theory of recursive functions and effective computability.* MIT Press, 1988.

[16] P. Valarcher. A complete characterization of intensional behaviours of primitive recursive algorithms. *Rapport de Recherche du LIR*, 96.11, 1996.

[17] P. Valarcher. Contribution à l'etude du comportement intentionel des algorithmes: le cas de la récursion primitive. *Thèse de doctorat, Université P 7*, 1996.

[18] P. Valarcher. Intensionality vs extensionality and primitive recursion. *ASIAN Computing Science Conference - LNCS*, 1179, 1996.

[19] J.E. Vuillemin. *Proof techniques for recursive programs.* PhD thesis, Standford, 1973.